Python Brain Teasers

EXERCISE YOUR MIND

```
1 class Player:
2     # Number of players in the Game
3     count = 0
4
5     def __init__(self, name):
6         self.name = name
7         self.count += 1
8
9
10 p1 = Player('Parzival')
11 print(Player.count)
```

What will this code print?

30 mind bending teasers & solutions

Miki Tebeka

Copyright

An Inside Job

inside.py

```
def add_n(items, n):
    items += range(n)


items = [1]
add_n(items, 3)
print(items)
```

Try to guess what the output is before moving to the next page.

This code will print: `[1, 0, 1, 2]`

In [Call Me Maybe] we talked about rebinding vs mutation. And most of the times `items += range(n)` is translated to `items = items + range(n)` which is rebinding.

There is a special optimization for `+=` in some cases. Here's what the documentation says (my emphasis):

> An augmented assignment expression like x += 1 can be rewritten as x = x + 1 to achieve a similar, but not exactly equal effect. In the augmented version, x is only evaluated once. Also, **when possible, the actual operation is performed in-place, meaning that rather than creating a new object and assigning that to the target, the old object is modified instead**.

A type defines how the `+` operator behaves with the `__add__` special method and can define `__iadd__` as a special case for `+=`. The documentation says:

> These methods are called to implement the augmented arithmetic assignments (`+=, -=, =, @=, /=, //=, %=, *=, <<=, >>=, &=, ^=, |=`). These methods should attempt to do the operation in-place (modifying self) and return the result (which could be, but does not have to be, self). If a specific method is not defined, the augmented assignment falls back to the normal methods.

The built-in `list` object defines `__iadd__` which calls the extend method.

What will happen if you change the code inside `add_n` to `items = items + range(n)`? You will get an exception: `TypeError: can only concatenate list (not "range") to list`.

In Python 3, range returns a `range` object.[1] Even though it *looks* like a `list` (`len`, `[]` and friends will work) you can't add it to a `list`.

If you want the rebinding code to work, you'll need to write `items = items + list(range(n))` and then the output will be `[1]`.

As a general rule, try not to mutate the object passed to your functions. This style of programming is called functional programming. Functional code is easier to test and reason about, give it a try - it's fun.

Further Reading

- Functional programming on Wikipedia
- Built-in range documentation
- Augmented assignment statements in the Python reference

- Functional Programming HOWTO in the Python documentation

[1] In Python 2 it returns a list.